Decomposition and Local Search Based Methods for the Traveling Umpire Problem

Tony Wauters[a,*], Sam Van Malderen[a], Greet Vanden Berghe[a]

[a] *CODeS, Department of Computer Science, KU Leuven, Gebroeders De Smetstraat 1, 9000 Gent, Belgium.*

Abstract

The Traveling Umpire Problem (TUP) is a challenging combinatorial optimization problem based on scheduling umpires for Major League Baseball. The TUP aims at assigning umpire crews to the games of a fixed tournament, minimizing the travel distance of the umpires. The present paper introduces two complementary heuristic solution approaches for the TUP. A new method called enhanced iterative deepening search with leaf node improvements (IDLI) generates schedules in several stages by subsequently considering parts of the problem. The second approach is a custom iterated local search algorithm (ILS) with a step counting hill climbing acceptance criterion. IDLI generates new best solutions for many small and medium sized benchmark instances. ILS produces significant improvements for the largest benchmark instances. In addition, the article introduces a new decomposition methodology for generating lower bounds, which improves all known lower bounds for the benchmark instances.

Keywords: OR in sports, Traveling Umpire Problem, heuristics, iterated local search, decomposition

1. Introduction and Problem Description

Sports scheduling enjoys an ever increasing interest of the operations research community. The focus lies mainly on scheduling the games of a competition or tournament. Scheduling tournaments turns out to be a challenging task: a small number of participants and rounds results in a large number of possible combinations, while the fraction of desired solutions is small due to context specific constraints and objectives. To master this complexity, researchers apply a wide range of combinatorial optimization techniques.

Sports scheduling, be it to a lesser extent, also considers assigning officials to the games in a tournament for tennis (Farmer et al. (2007)), football (Alarcón

*Corresponding author. Postal address: CODeS, Department of Computer Science, KU Leuven, Gebroeders De Smetstraat 1, 9000 Gent, Belgium. Phone: +3292658704

Email address: tony.wauters@cs.kuleuven.be (Tony Wauters)

et al. (to appear)), cricket (Wright (1991)), etc. Duarte et al. (2007a,b) introduce the Referee Assignment Problem (RAP), which considers the assignment of a number of referees with different qualifications to the games in a fixed tournament. In an area other than sports, Lamghari and Ferland (2011) consider the assignment of judges for the John Molson International Case Competition. Kendall et al. (2010) give a complete reference of the current state of sports scheduling in operations research.

The present paper focuses on the Traveling Umpire Problem (TUP), which is an academic version of the real world Major League Baseball umpire scheduling problem (MLB-USP). Trick and Yildiz (2007) introduce the TUP and describe it in more detail later (Trick et al. (2012)), comparing the problem to the MLB-USP (Evans (1988)). MLB-USP defines the rules and regulations imposed by the baseball league and umpire union for assigning 17 umpire crews, each consisting of four umpires, to cover all 780 series of an MLB tournament. Each series contains two up to four consecutive games between the same two teams out of all 30 teams. Even though only taking into account the most important constraints, the academic problem retains the most important characteristics of the real world umpire scheduling problem.

The TUP is related to the Traveling Tournament Problem (TTP, Easton et al. (2001)). The latter aims at finding a double round robin schedule for a season of Major League Baseball. Given $2n$ teams, the tournament consists of $4n-2$ rounds in which each team plays against exactly one other team in every round. The TUP considers assigning n umpire crews to the games in such a fixed TTP tournament. Its goal is obtaining a schedule which minimizes the travel distance of the umpire crews, while taking into account the following constraints:

C1. Every game in the tournament is officiated by exactly one umpire crew.
C2. An umpire crew officiates exactly one game per round.
C3. Every umpire crew should visit the home of every team at least once.
C4. An umpire crew must wait $q_1 - 1$ rounds before revisiting a team's home.
C5. An umpire crew must wait $q_2 - 1$ rounds before officiating the same team again.

With

$$q_1 = n - d_1 \quad (1)$$

$$q_2 = \lfloor \frac{n}{2} \rfloor - d_2 \quad (2)$$

whereby the values for parameters d_1 and d_2 range from 0 to n and 0 to $\lfloor \frac{n}{2} \rfloor$, respectively. Higher q values make the problem more constrained and drastically reduce the number of feasible solutions. Yildiz (2008) discusses the effect of different values of parameters q_1 and q_2 on the feasibility of the problem.

For simplicity, the above constraints and the remainder of this paper refer to umpire crews as a single umpire since a crew stays together throughout the whole season.

Although the complexity of the TUP is still open at the time of writing, the problem appears to be hard to solve. Minimizing the travel distance puts pressure on constraint C4 and C5 whereas enforcing constraints C4 and C5 increases the travel distance. Moreover, the assignment of one umpire influences the schedule of other umpires due to constraints C1 and C2. In addition, the problem description does not make a distinction between the different umpires. However, symmetrical solutions can be avoided by fixing the umpire assignments within a certain round. Trick et al. (2012) mention that the TUP can be seen as a special case of the vehicle routing problem with time windows.

The present paper presents two complementary heuristic approaches to the Traveling Umpire Problem. Enhanced iterative deepening search with leaf node improvements (IDLI) generates all partial schedules for a window of W rounds. The algorithm then greedily picks the best partial schedules to complete in subsequent stages. The second approach is a custom iterated local search algorithm (ILS) with a step counting hill climbing acceptance criterion. A steepest descent algorithm ensures all solutions are local optima before the ILS invokes the acceptance criterion. Finally, the article introduces a new decomposition methodology for generating tight lower bounds.

The structure of this paper is as follows: Section 2 presents an overview of the existing approaches to the TUP. Section 3 introduces the solution strategies and lower bound methodology. Section 4 reports and discusses computational results for benchmark instances. The last section summarizes conclusions and presents pointers for future research.

2. Related Research

Trick and Yildiz (2007) introduce TUP and formulate it as an Integer Program (IP) and a Constraint Program (CP). The same paper presents a greedy matching heuristic (GMH) with Bender's based modifications (GBNS). GBNS constructs a solution one round at a time by matching the umpires to games within the considered round. If the matching heuristic does not obtain a feasible matching at a certain round, GBNS examines the cause of the infeasibility and generates Benders' cuts. A very large neighborhood search algorithm then uses the Benders' cuts to resolve the infeasibility. The paper also tests the performance of the IP and CP formulations on benchmark instances and compares the results to those obtained by GBNS. The IP and CP formulation obtain solutions for relaxations of the benchmark instances, with $q_1 < n$ and/or $q_2 < \lfloor \frac{n}{2} \rfloor$. GBNS obtains solutions of higher quality in a shorter timespan, even for the most constrained versions of the benchmark instances.

Trick et al. (2012) present a simulated annealing algorithm with k-umpire neighborhood to both MLB-USP and TUP. The initial solution is constructed using GMH. The MLB-USP version of the algorithm obtains solutions of much higher quality than those constructed manually in previous years. The TUP version is capable of generating feasible solutions for relaxations of the larger benchmark instances.

Trick and Yildiz (2011) re-evaluate the performance of the algorithms presented by Trick and Yildiz (2007) on new benchmark instances. GBNS obtains the best results for a majority of the benchmark instances. New solver settings improve the performance of the CP and IP formulation.

Trick and Yildiz (2012) propose a genetic algorithm (GA) with a locally optimized crossover operator. Given two schedules and a round as crossover point, the crossover operator matches the rounds appearing before the crossover point in the first schedule to those appearing after the crossover point in the second schedule. The GA improves several of the best results for the TUP benchmark instances compared to the results obtained by Trick et al. (2012).

de Oliveira et al. (2013) strengthen the original IP formulation of Trick and Yildiz (2007) by removing one of the variables and some redundant constraints and by adding new valid inequalities. This noteworthy formulation improves all known lower bounds and is the first one capable of obtaining lower bounds for the larger problem instances. A relax-and-fix heuristic then uses this formulation to obtain solutions for the TUP. The relax-and-fix heuristic improves all best known solutions for the benchmark instances.

3. Approaches

The following sections present two new approaches to the TUP. Before going into detail, the first section clarifies the choice behind the final approaches. The last section discusses a methodology for generating tight lower bounds for the TUP.

3.1. Exploratory Experiments

Initial experiments have been conducted using both branch and bound and local search. The branch and bound algorithm assigns umpires round by round to the games of the tournament. It reassigns previous umpires when no feasible assignment for an umpire has been found. The local search algorithm improves a given initial solution by randomly exchanging umpire assignments within a given round. It explores infeasible solutions by adding violations of the hard constraints as a penalty term to the objective function. Figure 1 summarizes the results of initial experiments. It shows the performance of both algorithms as a function of the level of relaxation and the problem size.

Branch and bound performs well on small, non-relaxed benchmark instances (where $q_1 = n$ and $q_2 = \lfloor \frac{n}{2} \rfloor$). It is capable of pruning the search tree efficiently when the parameter values q_1 and q_2 of constraints C4 and C5 are large, compared to the total number of rounds. However, its ability to obtain solutions within reasonable time decreases rapidly with increasing problem size and when relaxing the problem instances by decreasing the values of q_1 and q_2.

The local search algorithm stops in local optima that are infeasible with respect to constraints C4 and C5 when parameter values q_1 and q_2 are large. When relaxing the problem instances by decreasing the parameter values, the performance of the local search algorithm increases, independently of the problem size.

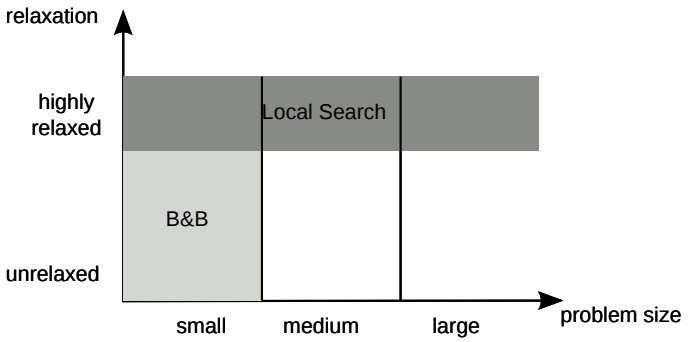


Figure 1: Summary of the performance of both the initial local search and the branch and bound algorithm on benchmark instances as a function of the problem size and the relaxation level of constraints C4 and C5.

Section 3.2 proposes a strategy to increase the performance of the initial branch and bound algorithm for larger problem instances, while retaining its performance for highly constrained versions of the instances. Section 3.3 improves the performance of the local search algorithm for more constrained versions of the instances.

3.2. Enhanced Iterative Deepening Search with Leaf Node Improvements

The branch and bound algorithm from Section 3 is extended along the lines of iterative deepening search. It is therefore called enhanced iterative deepening search with leaf node improvements (IDLI). Figure 2 shows a trace of the algorithm for a small problem instance. Algorithm 1 lists the pseudo-code.

Instead of assigning an umpire to every game in every round for the entire season of $4n-2$ rounds, IDLI decomposes the problem into windows of $W \leq 4n-2$ rounds each. The SOLVE-WINDOW procedure starts by generating all possible solutions within the first window of rounds using branch and bound. This results in a list of partial feasible schedules, which are consecutively sorted by total travel distance. The partial solutions are feasible if constraints C1, C2, C4 and C5 are satisfied. Constraints C4 and C5 are respected at the window boundaries by limiting the possible assignments in the subsequent window, based on the assignments in the partial solution. If an umpire is assigned to a team (home location) $X < q_2$ ($X < q_1$) rounds before the end of the window, the umpire cannot be reassigned to that team (home location) in the first $q_2 - X$ ($q_1 - X$) rounds of the next window. Constraint C3 is only enforceable once the algorithm obtains a complete schedule.

The next action to be performed depends on the last round of the current window. If the end round of the window is not equal to $4n-2$, the schedules in the sorted list are partial schedules. The S best of these partial schedules will be completed recursively by invoking the SOLVE-WINDOW procedure, whereby S is called the sample size.

The schedules in the sorted list are complete solutions for the given problem instance if the last round of the current window equals $4n-2$. The procedure

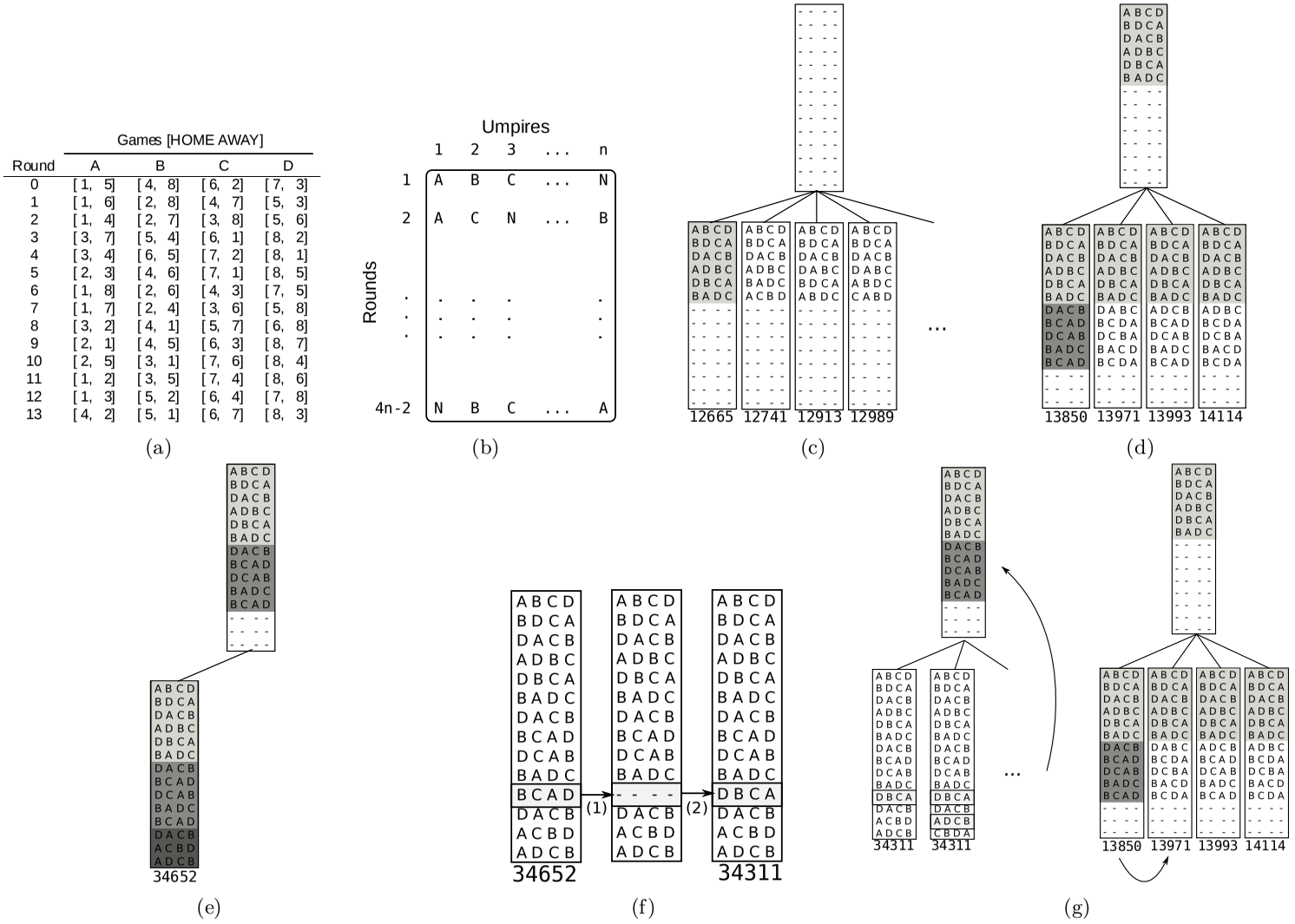


Figure 2: Graphical representation of IDLI for benchmark instance `umps8`. Figure (a) shows a tournament for the problem instance. For visualization purposes, the games of every round have been labeled with an index (A, B, C and D). A solution (Figure (b)) consists of a matrix in which the rows represent the rounds and the columns represent the umpires. By assigning labels to the cells of the matrix, the algorithm assigns the corresponding game of the tournament to an umpire. Starting from an empty solution and using a branch and bound algorithm, IDLI generates all feasible partial schedules for a window of W rounds and sorts them by objective function value (c). IDLI then picks the best schedule and completes its next windows recursively, fixing the assignments of the previous windows (d, e). Once a complete schedule has been obtained, a steepest descent algorithm with assignment neighborhood improves the complete schedule (f). When all possible complete solutions have been generated starting from the current partial schedule, IDLI returns to the previous window and recursively completes the remaining $S-1$ best partial schedules (g).

Algorithm 1 Pseudocode IDLI

```
Global: Solution best
Global: W                                                          ▷ Window size
Global: S                                                          ▷ Sample size
Global: I_i                                                        ▷ Improvement interval
Global: T_i                                                        ▷ Improvement time
Global: l          ▷ number of feasible solutions since last improvement, initially 0
Global: Solution temp                ▷ best feasible solution since last improvement

procedure SOLVE-WINDOW(Problem p, Solution partialSolution, start, end)

    if start == 0 then
        solutions ← Generate all feasible solutions for first window using branch and bound
    else
        solutions ← Generate S best feasible solutions for current window (rounds start
                    to end) using branch and bound while fixing partialSolution in previous
                    rounds, enforcing constraints C4 and C5 through the window boundaries
    end if

    solutions ← sort solutions by distance

    if solutions are complete solutions then                       ▷ when end = 4n − 2
        for solution ∈ solutions do
            if solution is feasible then

                solution ← perform steepest descent on solution
                if DISTANCE(best) > DISTANCE(solution) then
                    best ← solution
                    best ← IMPROVE(solution)
                else                                   ▷ Solution does not improve best known
                    if l = 0 or if DISTANCE(temp) > DISTANCE(solution) then
                        temp ← solution     ▷ Remember best unaccepted solution in interval
                    end if
                    l ← l + 1
                    if l == I_i then    ▷ Improve best out of I_i feasible, unaccepted solutions
                        temp ← IMPROVE(temp)
                        if DISTANCE(best) > DISTANCE(temp) then
                            best ← temp
                        end if
                        l ← 0
                    end if
                end if
            end if
        end for
    else                                ▷ solution are partial solution when end < 4n − 2
        for solution ∈ solutions do
            SOLVE-WINDOW(p, solution, end-1, end+W)                ▷ Solve subsequent window
        end for
    end if
    return best
end procedure
```

evaluates whether constraint C3 is violated or not. If not, IDLI executes the steepest descent (Section 3.2.1) algorithm. It accepts the resulting solution as new best solution when the total travel distance is smaller than that of the

current best solution. Once a solution has been accepted, an improvement procedure (Section 3.2.2) further improves the schedule.

IDLI invokes the same improvement procedure for the best solution out of I_i unaccepted complete and feasible solutions to enlarge the sample set of solutions eligible to improvement. I_i is called the improvement interval.

3.2.1. Steepest Descent Algorithm

The steepest descent solves a perfect matching to construct the neighbors of a given solution. The matching is similar to that used in the greedy matching heuristic presented by Trick and Yildiz (2007). It returns the optimal assignment for all the umpires within a single round, while fixing the assignments of the umpires in the other rounds. The cost of assigning an umpire to a game A in the chosen round X is a weighted sum of (1) the travel distances from the umpires' assignment in round $X-1$ to the location of A and the travel distance from A to its assignment in round $X+1$ and (2) the number of violated hard constraints, multiplied by a high value. The matching returns one assignment combination per round, which can be equal to the current assignment combination. Hence, the neighborhood consists of $max(4n-2)$ neighbors. Once all possible neighbors have been generated, steepest descent selects the neighbor for which the objective function value of the solution improves most. The algorithm stops and returns the best solution found when no further improvement can be obtained.

3.2.2. Improvement Procedure

The improvement procedure is a composition of two regular branch and bound algorithms. A forward branch and bound algorithm exploits the search space by considering alternative assignments in the last few rounds, keeping the first part of the schedule equal to that of the initial solution. A backward branch and bound algorithm performs the search in the other direction, exploiting the search space in the starting rounds of the schedule.

When IDLI invokes the improvement procedure, the backward branch and bound algorithm improves the solution for time T_i, the improvement time. Afterwards, the forward branch and bound algorithm starts improving the resulting solution for time T_i. This process is repeated until both the forward and backward branch and bound algorithm are incapable of further improving the resulting solution.

3.3. Iterated Local Search

The initial local search algorithm described in Section 3.1 is incapable of obtaining feasible solutions to more constrained problem instance versions. An iterated local search algorithm (ILS, Lourenço et al. (2003)) has been developed to cope with this characteristic. ILS is a hybrid single-solution based metaheuristic in which inner local search algorithms improve the current solution before applying the acceptance criterion of an outer metaheuristic.

The proposed ILS uses a step counting hill climbing (SCHC, Bykov and Petrovic (2013)) outer metaheuristic, nested with the steepest descent algorithm

Algorithm 2 Pseudocode ILS

```
1: procedure ITERATED-LOCAL-SEARCH(Problem p)
2:     S_0 ← CONSTRUCT-RANDOM-SOLUTION(p)
3:     best ← STEEPEST-DESCENT(S_0)
4:     current ← best
5:     threshold ← current
6:     while stopcriterion not met do
7:         temp ← PERTURB(current)
8:         temp ← STEEPEST-DESCENT(temp)
9:         current ← ACCEPTANCE(temp, threshold)
10:        if DISTANCE(current) < DISTANCE(best) then
11:            best ← current
12:        end if
13:        threshold ← UPDATE-THRESHOLD(current)
14:    end while
15:    return best
16: end procedure
```

described in Section 3.2. Algorithm 2 presents a high level structure of the ILS discussed in this section.

SCHC starts with a threshold equal to that of the objective value of an initial, randomly constructed, solution. It accepts every solution with an objective function value of higher quality than that of the current threshold. After a step of S iterations, SCHC updates the threshold with the objective function value of the current solution. The acceptance criterion is easily applicable because it consists of only a single parameter, the step size S.

To leave the local optimum in which the current solution resides, the PERTURB procedure of the ILS sometimes performs fairly large perturbations of the schedule using an exchange move operator. The exchange move operator chooses a number of random rounds in the schedule. In every round, two umpires are chosen at random and their assignments are swapped. After applying the exchange move, the ILS improves the solution using the steepest descent algorithm. It ascertains that the result is a local optimum with respect to the assignment neighborhood before invoking the acceptance criterion of the SCHC.

The running time of the steepest descent algorithm increases with the number of rounds considered by the perturbation procedure. The PERTURB procedure limits the number of rounds in a single perturbation to 12 in order to restrict the computational time. Moreover, the probability of considering i rounds in a single perturbation is given by the formula: $p(i) = \frac{12-(i-1)}{\sum_{j=1}^{12}(12-(j-1))}$.

3.3.1. Delta Evaluation

The neighborhood structures are not too complex. Therefore, the evaluation of solutions is the most time consuming part of the ILS. A full evaluation of a schedule takes time $\mathcal{O}(n^2 max(q_1, q_2)))$ because the evaluation function needs to iterate over n umpires for $4n - 2$ rounds, looking $max(q_1, q_2)$ rounds ahead due to constraints C4 and C5. The ILS applies delta evaluation rather than evaluating the complete schedule after performing a move (Talbi, 2009). Delta

evaluation takes advantage of two facts: (1) every move can be decomposed into a set of individual umpires for which the assignment changes, and (2) provided the original move does not violate constraint C1 and C2, an assignment change for a single umpire does not affect other umpires. Only the parts of the schedule that are affected by a move operation need to be re-evaluated.

To be able to benefit from delta evaluation, the ILS stores a state of the current schedule. The state of the schedule consists of (1) the distance traveled by each umpire throughout the complete tournament (2) the number violations for C3, C4 and C5 (3) the number of times every umpire has visited each team and (4) the number of times each umpire has visited a certain location (team) in the $q1$ ($q2$) rounds previous to every round. Every change in assignment removes the previous assignment from this state and adds the new one. If, for instance, the assignment of an umpire U in round X changes from game A to game B, the evaluation function first subtracts the assignment of U to A from the state in the following manner:

- it subtracts the distance from the location of U in round $X-1$ to the location of game A together with the distance from the location of game A to the location of U in round $X+1$ from the travel distance of U.
- it decrements the number of times U has visited the location at which game A is held. A resulting value equal to zero means that U does not visit this location in any of the rounds of the tournament anymore. If this is the case, the number of violations corresponding to constraint C3 is incremented.
- it decrements the number of times U has visited the location at which game A is held for round X up to round $X+q_1$. A resulting value equal to one for all rounds means that U does not violate constraint C4 for that location and round anymore. When this happens, the number of violations corresponding to constraint C4 is decremented.
- it decrements the number of times U has officiated the teams playing game A for round X up to round $X+q_2$. A resulting value equal to one for all rounds means that U does not violate constraint C5 for that round anymore. If this is the case, the number of violations corresponding to constraint C5 is decremented.

Afterwards, game B is added to the state in the opposite manner. Finally, the state contains new values for the travel distance and the number of violated constraints.

The evaluation function now takes into account one umpire at a time, for maximum $max(q_1, q_2)$ rounds due to constraints C4 and C5. This results in an evaluation time of $\mathcal{O}(max(q_1, q_2))$ per modified assignment. Delta evaluation results in a significant decrease in evaluation time as most moves result in only a few assignment modifications compared to the $\mathcal{O}(n^2)$ variables considered previously.

3.4. Lower Bound Methodology: a Decomposition Approach

Neither the ILS nor IDLI provide optimality gap information for the obtained solutions. In addition, the approaches reported in the literature do not obtain tight lower bounds for problem instances with large values of n. The current section presents a methodology for generating tight lower bounds, based on the decomposition of the problem into sub-problems. Algorithm 3 shows the generic structure of this lower bound methodology.

Algorithm 3 Generic structure for the lower bound methodology

```
1: procedure LOWERBOUND-DECOMPOSITION(Problem p, Window size W)
2:     start ← 0
3:     end ← 0
4:     LB ← 0
5:     while end < 4n − 2 do
6:         end ← max(start + W, 4n − 2)
7:         LB ← LB + CALCULATE-OPTIMAL(p, start, end)
8:         start ← end - 1
9:     end while
10:    return LB
11: end procedure
```

Every sub-problem consists of a window of $W \leq 4n - 2$ rounds. The CALCULATE-OPTIMAL method subsequently solves the resulting windows to optimality, without considering constraint C3. The CALCULATE-OPTIMAL procedure represents any method capable of solving sub-problems to optimality. A modified version of the formulation presented by de Oliveira et al. (2013) is used to solve the windows to optimality. It only takes into account variables within the considered rounds of the window while the middle round of the sub-problem is used as symmetry breaking round.

The first round of every window, apart from the initial window, is the same as the last round of the previous window, ensuring that the resulting lower bound includes the travel distance between windows. The requirement that the assignments in this overlapping round have to be equal is relaxed, making the procedure parallelizable. This relaxation does not influence the validity of the lower bound.

The sum of the travel distance within the windows is a valid lower bound for the TUP because (1) the optimality of the sub-problems solutions ensures that no smaller distance is obtainable within the window, (2) the procedure relaxes constraints C4 and C5 at the start of a new window and (3) the methodology does not consider constraint C3.

4. Computational results

The current section reports the performance of the algorithms on benchmark instances for the TUP. Trick and Yildiz (2013) provide a data set consisting of several problem instances, with size ranging from 2 to 16 umpires. The instances

contain a TTP tournament and a distance matrix. The tournament specifies the competing teams and locations of the games in every round. The distance matrix provides the distance between these locations. For each problem size, the data set contains several instances with equal tournaments while providing a permutation of the original distance matrix. The instances names contain the number of teams followed by a letter if the distance matrix is a permutation of the original one.

In what follows, benchmark instances with $n \leq 5$ are referred to as "small", instances with $5 < n \leq 10$ are denoted as "medium sized" and those with $n > 10$ are called "large". Variations of the problem instances can be constructed by changing the values of q_1 and q_2. Opting for smaller values for q_1 (q_2) means that an umpire has to wait a smaller number of rounds in between revisiting the same location (team).

All procedures have been implemented using the Java$^{\text{TM}}$ programming language version 1.7. The results of the heuristics are generated using a single Intel Xeon E5-2670 CPU thread running at 2.6GHz.

4.1. Lower bounds

Table 1 shows the results obtained by the lower bound methodology for small problem instances with at most 10 teams. The decomposition methodology obtains lower bounds equal to the optimal values for every instance with at most 8 teams, apart from instance `8A`.

Larger window sizes do not necessarily result in better lower bounds. A possible explanation for this phenomenon is that for some window sizes, the decomposition method is incapable of dividing the rounds equally among the windows. The last window ends up with the remainder of the rounds. At a certain point, the last window becomes too small for constraints C4 and C5 to have much influence on the travel distance, resulting in a smaller total travel distance.

Table 2 shows the results obtained by the lower bound methodology for instances with more than 10 teams, for which no optimal values are available in the literature. The table compares the results with the current best lower bounds obtained by de Oliveira et al. (2013) ($\mathcal{F}$ Best) and those originally obtained by Trick and Yildiz (2013) (TYi). The 14-team instances are solved up to window size $W = 15$ and the 16-team instances up to window size $W = 12$ or until the computation time exceeded the available time. The $LB3$ column lists the lower bounds obtained by the decomposition method within three hours of calculation time. The $LB+$ column shows the best bounds obtained in more than three hours of calculation time. The WS-MAX column lists the maximum considered window size.

The decomposition method improves all known lower bounds for the larger problem instances. The last column shows that the bound improves with increasing window sizes. The window size solvable within the given timespan decreases for more constrained versions of the same problem instance.

The table lists all lower bounds next to the problem instance version for which the respective methods obtained them. The following conjecture states

				Window size																
	Instance	q_1	q_2	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Lower bound	4*	2	1	**5176**	**5176**	**5176**	**5176**	**5176**	-	-	-	-	-	-	-	-	-	-	-	-
	6*	3	1	13971	**14077**	**14077**	**14077**	**14077**	**14077**	**14077**	**14077**	**14077**	-	-	-	-	-	-	-	-
	6A*	3	1	12919	15351	**15457**	**15457**	13605	**15457**	**15457**	**15457**	**15457**	-	-	-	-	-	-	-	-
	6B*	3	1	14607	15003	**16716**	15828	16565	**16716**	**16716**	15828	**16716**	-	-	-	-	-	-	-	-
	6C*	3	1	**14396**	**14396**	**14396**	**14396**	**14396**	**14396**	**14396**	**14396**	**14396**	-	-	-	-	-	-	-	-
	8*	4	2	33651	33924	33723	33942	34293	**34311**	**34311**	33942	33723	34293	34257	**34311**	**34311**	-	-	-	-
	8A	4	2	29895	30419	30660	30901	31436	31251	31266	30901	30857	31436	31347	**31448**	**31448**	-	-	-	-
	8B*	4	2	31707	32324	31827	32619	32663	32436	32617	32619	32122	**32731**	32727	**32731**	**32731**	-	-	-	-
	8C*	4	2	27420	27922	28548	28670	29476	29548	29253	28670	28879	29476	**29879**	**29879**	**29879**	-	-	-	-
	10	5	2	47720	47951	48066	48098	48297	48255	48521	48740	**48759**	**48759**	48579	48600	48599	48718	48740	48740	**48759**
	10A	5	2	44443	44939	45236	45743	45777	45587	45310	45815	**46059**	46040	45940	**46059**	45997	45763	45815	45815	**46059**
	10B	5	2	43823	44513	45091	45091	45091	45303	45175	45091	45091	45091	45048	45303	45331	45175	**45435**	**45435**	**45435**
	10C	5	2	39395	40918	41294	41769	41934	42103	42771	42253	43008	42807	42947	42947	42166	42903	42772	42543	**43120**
Time (ms)	4	2	1	57	27	16	16	11	-	-	-	-	-	-	-	-	-	-	-	-
	6	3	1	111	64	36	34	35	31	28	28	23	-	-	-	-	-	-	-	-
	6A	3	1	138	53	36	36	34	32	27	29	27	-	-	-	-	-	-	-	-
	6B	3	1	118	52	29	29	32	31	26	26	29	-	-	-	-	-	-	-	-
	6C	3	1	129	50	30	34	32	38	30	26	24	-	-	-	-	-	-	-	-
	8	4	2	213	86	108	100	67	69	86	69	65	89	70	75	69	-	-	-	-
	8A	4	2	194	89	90	71	73	72	83	80	69	62	75	73	72	-	-	-	-
	8B	4	2	215	89	79	73	69	71	81	68	68	68	86	84	70	-	-	-	-
	8C	4	2	220	89	88	79	82	71	87	68	61	70	77	71	70	-	-	-	-
	10	5	2	354	199	181	178	223	216	255	271	270	316	304	331	531	1335	1674	1496	1741
	10A	5	2	340	194	179	156	218	225	244	273	274	297	311	345	337	330	422	391	944
	10B	5	2	350	212	182	172	210	229	255	312	307	351	319	378	356	401	614	873	740
	10C	5	2	365	182	180	183	234	232	331	308	659	445	615	578	1020	1298	1582	2538	4012

Table 1: Lower bounds obtained by the decomposition methodology using different window sizes for small instances with $n \leq 5$. The * denotes that the method obtains a lower bound equal to the optimal value. The table lists the best values in bold.

Instance	q_1	q_2	TYi	$\mathcal{F}$ Best	$LB3$	WS	$LB+$	WS	WS-MAX
14	7	3	141253	150871	156536	13	**159797**	14	15
	6	3	141064	150041	**156551**	15	**156551**	15	15
	5	3	141134	150270	**153066**	14	**153066**	14	15
14A	7	3	133279	143517	151406	11	**153199**	14	15
	6	3	133194	143931	**150998**	14	**150998**	14	15
	5	3	133023	143504	**148299**	14	**148299**	14	15
14B	7	3	131373	142614	149910	11	**151059**	14	15
	6	3	130799	143378	**149267**	14	**149267**	14	15
	5	3	130628	143706	**147534**	14	**147534**	14	15
14C	7	3	126843	141268	151122	14	**151581**	15	15
	6	3	126613	141791	**148728**	14	**148728**	14	15
	5	3	126427	141801	**146764**	14	**146764**	14	15
16	8	4	134471	151748	168847	9	**185939**	11	11
	8	2	134347	143840	**151481**	12	**151481**	12	12
	7	3	121933	145987	155707	10	**158480**	11	12
	7	2	121670	141440	**147138**	11	**147138**	11	12
16A	8	4	148377	166626	**185119**	9	**185119**	9	9
	8	2	146992	157972	**162788**	12	**162788**	12	12
	7	3	137178	160314	170342	10	**172964**	12	12
	7	2	137806	155342	**161640**	12	**161640**	12	12
16B	8	4	146646	162251	188195	9	**208418**	11	11
	8	2	145058	158035	**167768**	12	**167768**	12	12
	7	3	139833	158244	170940	10	**173023**	12	12
	7	2	139742	155403	**164012**	12	**164012**	12	12
16C	8	4	145012	165431	179213	8	**188561**	10	10
	8	2	144398	160596	163543	10	**166001**	11	12
	7	3	142467	161838	170133	10	**171377**	11	12
	7	2	142399	158527	**163305**	11	**163305**	11	12
30	5	5	-	367877	403725	7	**413103**	11	11

Table 2: Lower bounds obtained by the decomposition methodology (columns $LB3$ and $LB+$) for instances with at least 14 teams compared to the known lower bounds in the literature (column $\mathcal{F}$ Best) and the initial bounds (column TYi).

that a lower bound for a relaxed versions of a problem instance is also a lower bound for more constrained versions of the instance.

Conjecture 4.1. *A valid lower bound for an instance with values q_1 and q_2 is also valid for the same instance with values $q_1' \geq q_1$ and $q_2' \geq q_2$.*

As a result, the lower bound for instance `14` with $q_1 = 6$ and $q_2 = 3$ can replace the lower bound for instance `14` with $q_1 = 7$ and $q_2 = 3$ in the $LB3$ column, resulting in a better bound.

4.2. IDLI

In what follows, the performance of the IDLI approach is assessed on the benchmark instances. Instead of generating all possible solutions for each window, the branch and bound algorithm stores sample size S solutions and prunes the search tree when the distance of the current schedule is larger than the distance of the worst solution in the list. The schedule's distance contains a lower

bound that corresponds to the shortest path for all umpires to travel from their current assignment to any game in the last round of the window.

The IDLI procedure consists of several parameters: the window and sample size and the improvement interval and time. IDLI obtains optimal solutions for every problem instance with at most 10 teams using a window size of 6, a sample size of n, an improvement interval of 1000 and an improvement time of 1 second. Table 3 shows the values and calculation times. For larger instances, the final parameter settings result from initial experiments discussed in the following section.

Instance	IDLI	Time (ms)
4	5176*	11
6	14077*	32
6A	15457*	43
6B	16716*	37
6C	14396*	40
8	34311*	47
8A	31490*	45
8B	32731*	61
8C	29879*	46
10	48942*	6243
10A	46551*	4195
10B	45609*	4328
10C	43149*	13217

Table 3: IDLI obtains optimal solutions for all instances with at most 10 teams.

4.2.1. Best parameter settings

The IDLI parameters for the larger benchmark instance have been chosen based on the result of a full factorial experiment for a selection of problem instances. Table 4 lists the levels of the factors that have been considered in the model, next to the corresponding problem instance.

A multi-way ANOVA test on the results of the full factorial experiment shows that only three parameters have significant influence on the final result. These parameters are the sample size, the window size and the improvement interval. The improvement time has no significant influence on the end result.

Figures 3a and 3c show which parameter values should be considered for the final benchmark on all instances. IDLI performs best when considering a large window size, sample size equal to n and a large improvement interval. A result similar to that of `14` has been obtained for the full factorial test on instance `14A`.

The test has also been conducted on more relaxed versions of the same problem instances. The improvement time is fixed to 1000 because the previous experiments show that the improvement time has no significant influence on the final result. Figure 3b shows the result for problem instance `14` with $q_1 = 5$ and $q_2 = 3$. Figure 3d shows the result for instance `16` with parameter values $q_1 = 7$

Instance	14, 14A	16
$q1$	7	7
$q2$	3	3
Window size	4, 5, 6, 7, 8	3, 4, 5, 6
Sample size	7, 35, 70, 700	8, 40, 80, 800
Improvement Interval	0, 1, 100, 1000	0, 1, 100, 1000
Improvement time (ms)	100, 1000, 5000	100, 1000, 5000

Table 4: Problem instances, factors and corresponding levels considered for a full factorial experiment

and $q_2 = 2$. The results are similar to those for the more constrained versions. However, IDLI was incapable of obtaining a feasible solution for the relaxation of the 16 instance with several parameter settings. For these parameter settings, the results have been substituted with an objective value larger than any of the obtained objective values to generate Figure 3d.

4.2.2. Benchmark results

IDLI is executed on the larger problem instances with an improvement time equal to 1 seconds, the improvement interval fixed to 1000 and n is used as sample size. The window size is fixed to 8 for the 14-team instances and 5 for the 16-team instances.

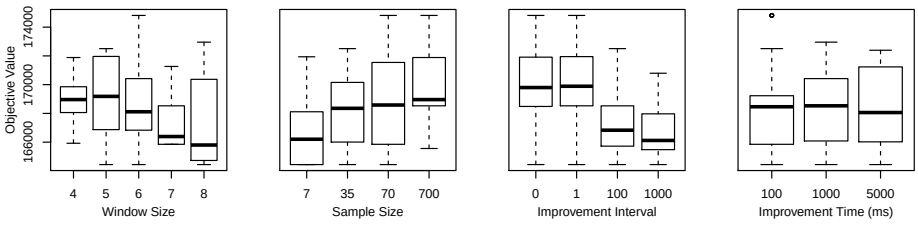
Table 5 shows the results obtained by IDLI within 3 hours of computation time, as well as the best known lower bound for different versions of each problem instance. The last column shows the optimality gap. IDLI obtains results which are close to the best known lower bounds with less than 11% optimality gap for all 14-team instances. In addition, IDLI obtains objective function values for the relaxations of the 16-team instances that are also close to the best known lower bounds. However, IDLI does not reach a solution to the most constrained version of the 16-team instances and the 30-team instance.

The table lists the objective function values next to the problem instance for which they are obtained. The following theorem, with proof, states that results for a relaxation of a problem instance can be substituted with the results for more constrained versions of the same problem instance.

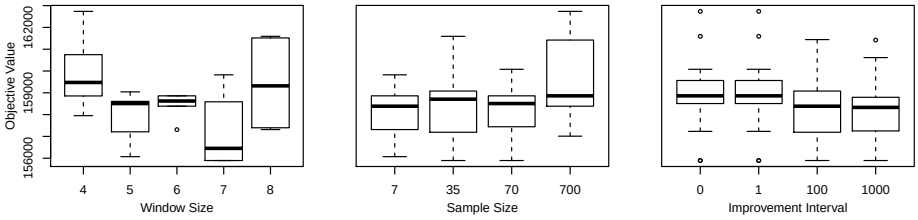
Theorem 4.1. *A feasible solution to an instance with parameter value q_1 for constraint C4 is also feasible for the same instance with parameter value $q_1' \leq q_1$.*

Proof. By contradiction. Let S be a feasible solution to a problem with parameter value q_1 for constraint C4. Consider, ceteris paribus, a constraint value $q_1' \leq q_1$. Assume S is infeasible with respect to constraint C4 for constraint value q_1'. This means that there exists an umpire in S who visits the same location more than once within q_1' consecutive rounds. Since $q_1' \leq q_1$ the same umpire also violates constraint C4 with parameter value q_1, making S infeasible for parameter value q_1.

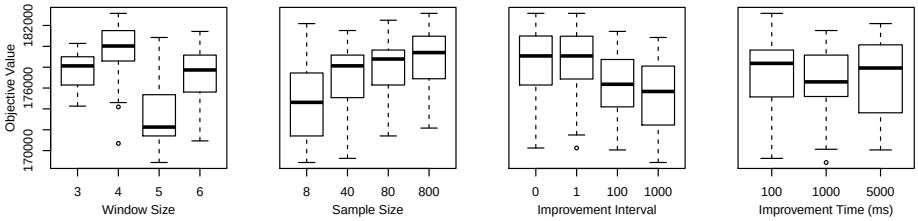
□



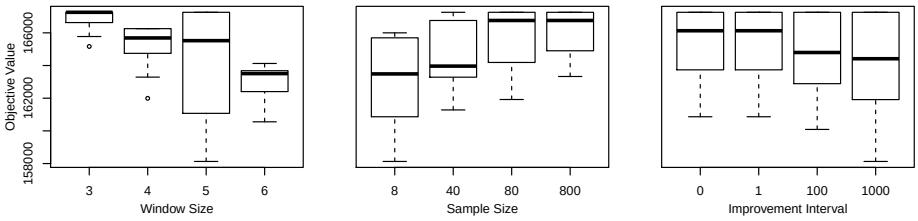
(a) Problem instance `14` with parameter values $q_1 = 7$ and $q_2 = 3$.



(b) Problem instance `14` with parameter values $q_1 = 5$ and $q_2 = 3$.



(c) Problem instance `16` with parameter values $q_1 = 7$ and $q_2 = 3$.



(d) Problem instance `16` with parameter values $q_1 = 7$ and $q_2 = 2$.

Figure 3: Results of a full factorial experiment for several problem instances.

The theorem is also applicable to constraint C5 with parameter values $q_2' \leq q_2$ and the combination of constraints C4 and C5 with parameter values $q_1' \leq q_1$ and $q_2' \leq q_2$. As a result, the objective function value for a relaxation of instance `14C` with parameter values $q_1 = 5$ and $q_2 = 3$ can be substituted with the results of the relaxation of instance `14C` with parameter values $q_1 = 6$ and $q_1 = 3$, resulting in a better objective function value.

Instance	q_1	q_2	LB+	IDLI	Gap [%]
14	7	3	159797	164440	2.82
	6	3	156551	159505	1.85
	5	3	153066	157314	2.7
14A	7	3	153199	159610	4.02
	6	3	150998	153216	1.45
	5	3	148299	150698	1.59
14B	7	3	151059	157884	4.32
	6	3	149267	152740	2.27
	5	3	147534	150637	2.06
14C	7	3	151581	157373	3.68
	6	3	148728	150986	1.5
	5	3	146764	151193	2.93
16	8	4	185939	-	-
	8	2	151481	168086	9.88
	7	3	158480	168860	6.15
	7	2	147138	158114	6.94
16A	8	4	185119	-	-
	8	2	162788	173728	6.3
	7	3	172964	181486	4.7
	7	2	161640	169158	4.44
16B	8	4	208418	-	-
	8	2	167768	181119	7.37
	7	3	173023	181565	4.7
	7	2	164012	171336	4.27
16C	8	4	188561	-	-
	8	2	166001	184806	10.18
	7	3	171377	184320	7.02
	7	2	163305	170805	4.39
30	5	5	413103	-	-

Table 5: Results obtained by IDLI for problem instances with at least 14 teams.

4.3. ILS

The ILS contains only a single parameter: the step size which is used by the acceptance criterion of SCHC. The initial experiments reveal the best step size settings for instances with at least 14 teams. The ILS obtains good solutions using any reasonable step size for instances with at most 10 teams. Table 6 shows the best, average and worst results obtained by ILS using a step size of

		ILS		
Instance	Optimal	Best	Average	Worst
4	5176	5176*	5176*	5176*
6	14077	14077*	14077*	14077*
6A	15457	15457*	15457*	15457*
6B	16716	16716*	16716*	16716*
6C	14396	14396*	14396*	14396*
8	34311	34311*	34311*	34311*
8A	31490	31490*	31490*	31490*
8B	32731	32731*	32731*	32731*
8C	29879	29879*	29879*	29879*
10	48942	48942*	49155	49626
10A	46551	46551*	46660	46867
10B	45609	45609*	45667	45705
10C	43149	43149*	43392	43525

Table 6: Results obtained by ILS for instances with at most 10 teams. Optimal values are marked with a '*'.

5000 on the strict versions of the small problem instances. The table reports the results for 6 runs of 10 minutes on each problem instance.

The ILS always obtains optimal objective function values for all instances up to the 8-team instances. It obtains the optimal objective function value at least once for all 10-team problem instances. The average objective function value for the 10-team instances is also close to the optimal value.

4.3.1. Parameter settings

Experiments in this section provide information concerning good step sizes for problem instances with at least 14 teams. Tables 7 and 8 show the average result for different window sizes obtained by ILS for 5 runs of 3 hours each per problem instance. The table indicates the average solution as infeasible when one of the results is infeasible. The step size has to be at least 15000 to obtain feasible solutions for the most constrained 14-team instances. The best step size appears to be instance specific for the 14-team instances. However, a step size of 20000 seems to yield the most promising results. A step size of 10000 yields good results for the 16-team instances.

			Step size				
Instance	q_1	q_2	10000	15000	20000	25000	30000
14	7	3	182255	180353	**175345**	180968	176962
14A	7	3	inf	176825	175554	179587	**174676**
14B	7	3	inf	178104	**173723**	177463	173860
14C	7	3	174893	177647	177036	**173016**	176936

Table 7: Average objective function values resulting from 6 runs using different step sizes on the most constrained 14-team instances.

			Step size		
Instance	q_1	q_2	10000	15000	20000
16	8	2	183886	**182130**	194234
16A	8	2	**193376**	197282	199396
16B	8	2	**199396**	200724	203106
16C	8	2	**193446**	194417	199039

Table 8: Average objective function values resulting from 6 runs using different step sizes on the 16-team instances.

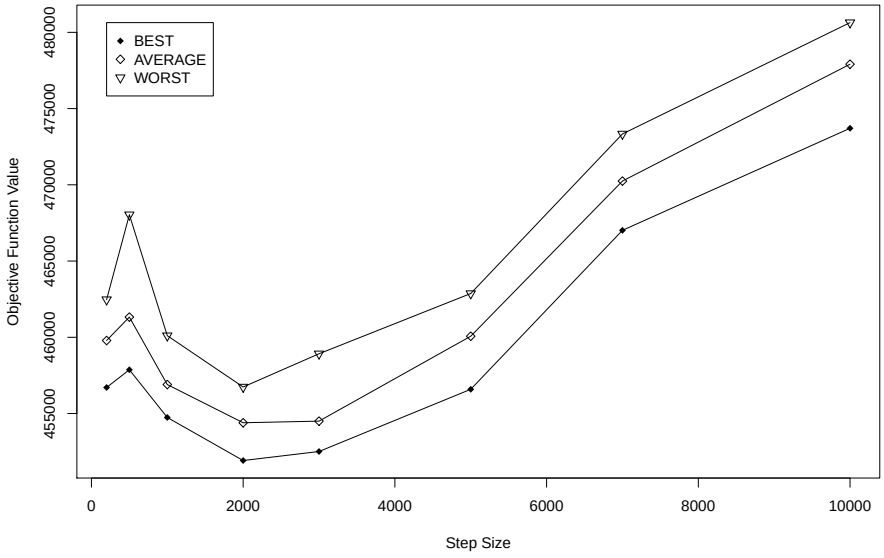


Figure 4: Best average and worst objective function values for relaxations of the 30-team instance, with $q_1 = 5$ and $q_2 = 5$

Figure 4 shows the best, average and worst results obtained by ILS for the 30-team instance with $q_1 = 5$ and $q_2 = 5$ over 6 runs. Contrary to the results for smaller instances, the figure clearly shows that step size 2000 yields the most promising results. This step size is used in benchmark experiments for instances with more than 16 teams.

4.3.2. Benchmark results

The ILS is executed 10 times on the large problem instances. Each run takes 3 hours. The step size is set equal to 20000 for the 14-team instances and 10000 for the 16-team instance. Table 9 shows the best, average and worst results obtained by IDLI for several versions of the 14-team and 16-team instances.

The worst column contains 'inf' followed by the number of infeasible solutions when one or more solutions are infeasible.

The algorithm shows good performance for relaxations of the 14 and 16-team problem instances by obtaining similar and sometimes better results than IDLI. It does not obtain a feasible solution for four of the most constrained problem instance versions in at most two out of ten runs.

				ILS		
Instance	q_1	q_2	IDLI	Best	Average	Worst
14	7	3	**164440**	172183	178312	187329
	6	3	**159505**	166014	169702	173213
	5	3	**157314**	158369	160309	162904
14A	7	3	**159610**	170040	179182	Inf (1)
	6	3	**153216**	161809	165031	168726
	5	3	**150698**	153645	156107	158569
14B	7	3	**157884**	167364	174150	183116
	6	3	**152740**	158364	162439	168896
	5	3	**150637**	151664	154260	156453
14C	7	3	**157373**	169707	175522	182255
	6	3	**150986**	155577	160374	165929
	5	3	151193	**150134**	153343	156552
16	8	4	-	-	-	-
	8	2	**168086**	175929	183412	189944
	7	3	**168860**	204227	210032	Inf (2)
	7	2	**158114**	169272	172417	175504
16A	8	4	-	-	-	-
	8	2	**173728**	186829	194218	200142
	7	3	**181486**	204227	210032	Inf (2)
	7	2	**169158**	169272	172417	175504
16B	8	4	-	-	-	-
	8	2	**181119**	193591	198786	207607
	7	3	**181565**	210990	215142	223794
	7	2	**171336**	173119	176581	180428
16C	8	4	-	-	-	-
	8	2	**184806**	186707	194643	199025
	7	3	**184320**	201509	211525	Inf (1)
	7	2	**170805**	173361	175796	178075

Table 9: Best, average and worst results obtained during 10 runs by the ILS for the 14-team and 16-team instances.

Table 10 shows the best, average and worst results obtained during 10 runs of 5 hours by the ILS for relaxations of problem instances with at least 26 teams. The step size is set equal to 2000. The $LB+$ column lists the lower bounds obtained by the decomposition methodology for the 26-team, 28-team

and 32-team instances with a window size equal to 9.

The table indicates that the main contribution of the ILS approach is its ability to obtain high quality solutions for relaxations of problem instances with at least 26 teams. The optimality gap between the average result and the best known lower bound for the 30-team instance is smaller than 10%.

				ILS		
Instance	q_1	q_2	$LB+$	Best	Average	Worst
26	5	5	318690	354134	362071	368453
28	5	5	358593	398101	401933	406773
30	5	5	413103	451917	455452	459520
32	5	5	443281	502890	508553	513268

Table 10: Best, average and worst results obtained in 10 runs by ILS for benchmark instances with at least 26 teams

4.4. Result summary

Table 11 summarizes the results obtained by the new heuristics (IDLI, ILS) and compares them to the initial results reported by Trick and Yildiz (2013) (TYi) and the results obtained by de Oliveira et al. (2013) (RF 4,6,7). The results were obtained within 3 hours of computation time for instances with at most 16 teams and 5 hours for larger problem instances. The best bounds column lists all bounds generated by the decomposition method.

IDLI improves all initial objective function values and improves 14 out of 24 solutions compared with RF 4,6,7. ILS also improves most initial objective function values for problem instances with at most 16 teams. ILS improves the best known result for the realistic problem instance with 30 teams.

The last two columns compare the best results obtained by the new heuristics over all different parameter settings to the overall best objective function values obtained by the relax-and-fix heuristic of de Oliveira et al. (2013). The table shows that both methods are very competitive. Both generate state of the art results for a similar number of problem instances.

5. Conclusion and Future Work

Two new and complementary approaches for the Traveling Umpire Problem were introduced. An enhanced iterative deepening search approach (IDLI) divides the problem into windows that are solved recursively. IDLI generates new best results for 14-team and 16-team benchmark instances. The second approach is a custom iterated local search (ILS) approach in which a steepest descent algorithm improves the current solutions of a step counting hill climbing metaheuristic. The ILS obtains good solutions for relaxations of the small and medium sized problem instances and generates high quality results for relaxations of problem instances with more than 16 teams as well.

In addition, a methodology has been proposed to obtain lower bounds for the Traveling Umpire Problem by decomposing the problem into sub-problems. The IP formulation of de Oliveira et al. (2013) has been used to solve the sub-problems. The proposed methodology improves all known lower bounds for benchmark instances.

Future research can be conducted to improve either one of the solution approaches. IDLI could benefit from a change in objective of the sorting procedure. Instead of completing only schedules with the smallest travel distance, other objectives can be incorporated. Such objectives could take into account the likelihood of the final solution being feasible. Replacing the branch and bound implementation with a more efficient method, capable of generating all solutions for a window, has the potential to yield good results for the larger problem instances. The ILS and other local search methods for the TUP will benefit from exploiting the characteristics of more sophisticated neighborhoods, which are generated based on the currently violated constraints.

			Lower Bounds			Objective Function Values					
Instance	q_1	q_2	TYi	$\mathcal{F}$ Best	LB+	TYi	RF 4,6,7	IDLI	ILS	RF Best	Our Best
14	7	3	141253	150871	**159797**	166964	168408	**164440**	172183	165573	**164440**
	6	3	141064	150041	**156551**	173681	160907	**159505**	166014	159522	**159505**
	5	3	141134	150270	**153066**	165558	**156456**	157314	158369	**155439**	155893
14A	7	3	133279	143517	**153199**	160407	163471	**159610**	170040	160046	**158760**
	6	3	133194	143931	**150998**	164857	156437	**153216**	161809	154628	**153216**
	5	3	133023	143504	**148299**	162380	**149331**	150698	153645	**149331**	150698
14B	7	3	131373	142614	**151059**	161129	**157884**	**157884**	167364	**157884**	**157884**
	6	3	130799	143378	**149267**	168476	153611	**152740**	158364	153611	**152740**
	5	3	130628	143706	**147534**	160443	**150268**	150637	151664	150268	**149621**
14C	7	3	126843	141268	**151581**	159461	160274	**157373**	169707	159518	**154913**
	6	3	126613	141791	**148728**	166395	152158	**150986**	155577	152158	**150858**
	5	3	126427	141801	**146764**	163662	150346	151193	**150134**	**149662**	150130
16	8	4	134471	151748	**185939**	-	-	-	-	-	-
	8	2	134347	143840	**151481**	178775	**160705**	168086	175929	**160705**	165638
	7	3	121933	145987	**158480**	185966	169994	**168860**	204227	169994	**168860**
	7	2	121670	141440	**147138**	166114	**155766**	158114	169272	**153978**	158114
16A	8	4	148377	166626	**185119**	-	-	-	-	-	-
	8	2	146992	157972	**162788**	188432	173950	**173728**	186829	173950	**172966**
	7	3	137178	160314	**172964**	199016	182889	**181486**	204227	181119	**179960**
	7	2	137806	155342	**161640**	172728	**164620**	169158	169272	**164620**	169158
16B	8	4	146646	162251	**208418**	-	-	-	-	-	-
	8	2	145058	158035	**167768**	201039	182673	**181119**	193591	182673	**180888**
	7	3	139833	158244	**173023**	202395	187488	**181565**	210990	187007	**181565**
	7	2	139742	155403	**164012**	184923	**170194**	171336	173119	**170194**	171336
16C	8	4	145012	165431	**188561**	-	-	-	-	-	-
	8	2	144398	160596	**166001**	202023	**180221**	184806	186707	**180221**	183554
	7	3	142467	161838	**171377**	213157	185528	**184320**	201509	185528	**184181**
	7	2	142399	158527	**163305**	181013	**169184**	170805	173361	**169184**	170805
30	5	5	-	367877	**413103**	483224	471724	-	**451917**	466765	**450919**

Table 11: Summary of the performance of the lower bound method, IDLI and the ILS, compared to the best known results in literature.

Alarcón, F., Durán, G., Guajardo, M.. Referee assignment in the chilean football league using integer programming and patterns. International Transactions in Operational Research to appear;.

Bykov, Y., Petrovic, S.. An initial study of a novel step counting hill climbing heuristic applied to timetabling problems. In: Kendall, G., Vanden Berghe, G., McCollum, B., editors. Proceedings of the 6th Multidisciplinary International Scheduling Conference: Theory & Applications (MISTA). Gent, Belgium; 2013. .

Duarte, A., Ribeiro, C., Urrutia, S.. A hybrid ils heuristic to the referee assignment problem with an embedded mip strategy. In: Bartz-Beielstein, T., Blesa Aguilera, M., Blum, C., Naujoks, B., Roli, A., Rudolph, G., Sampels, M., editors. Hybrid Metaheuristics. Springer Berlin Heidelberg; volume 4771 of *Lecture Notes in Computer Science*; 2007a. p. 82–95.

Duarte, A., Ribeiro, C., Urrutia, S., Haeusler, E.. Referee assignment in sports leagues. In: Burke, E.K., Rudová, H., editors. Practice and Theory of Automated Timetabling VI. Springer Berlin Heidelberg; volume 3867 of *Lecture Notes in Computer Science*; 2007b. p. 158–173.

Easton, K., Nemhauser, G., Trick, M.. The Traveling Tournament Problem Description and Benchmarks, 2001.

Evans, J.R.. A microcomputer-based decision support system for scheduling umpires in the american baseball league. Interfaces 1988;18(6):42 – 51.

Farmer, A., Smith, J.S., Miller, L.T.. Scheduling umpire crews for professional tennis tournaments. Interfaces 2007;37(2):187–196.

Kendall, G., Knust, S., Ribeiro, C.C., Urrutia, S.. Scheduling in sports: An annotated bibliography. Computers & Operations Research 2010;37(1):1 – 19.

Lamghari, A., Ferland, J.A.. Assigning judges to competitions of several rounds using tabu search. European Journal of Operational Research 2011;210(3):694 – 705.

Lourenço, H.R., Martin, O.C., Stützle, T.. Iterated local search. In: Glover, F., Kochenberger, G., editors. Handbook of Metaheuristics. Springer US; volume 57 of *International Series in Operations Research & Management Science*; 2003. p. 320–353.

de Oliveira, L., de Souza, C.C., Yunes, T.. Improved bounds for the traveling umpire problem: A stronger formulation and a relax-and-fix heuristic. European Journal of Operational Research 2013;In press.

Talbi, E.G.. Metaheuristics: From Design to Implementation. Wiley Publishing, 2009.

Trick, M.A., Yildiz, H.. Bender's cuts guided large neighborhood search for the traveling umpire problem. In: Hentenryck, P.V., Wolsey, L., editors. Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. Springer Berlin Heidelberg; number 4510 in Lecture Notes in Computer Science; 2007. p. 332–345.

Trick, M.A., Yildiz, H.. Benders' cuts guided large neighborhood search for the traveling umpire problem. Naval Research Logistics (NRL) 2011;58(8):771 – 781.

Trick, M.A., Yildiz, H.. Locally optimized crossover for the traveling umpire problem. European Journal of Operational Research 2012;216(2):286 – 292.

Trick, M.A., Yildiz, H.. Traveling umpire problem, benchmark instances. 2013. URL: `http://mat.gsia.cmu.edu/TUP/`.

Trick, M.A., Yildiz, H., Yunes, T.. Scheduling major league baseball umpires and the traveling umpire problem. Interfaces 2012;42:232 – 244.

Wright, M.B.. Scheduling english cricket umpires. The Journal of the Operational Research Society 1991;42(6):pp. 447–452.

Yildiz, H.. Methodologies and Applications for Scheduling, Routing & Related Problems. Ph.D. thesis; Carnegie Mellon University; 2008.